EM – Electronic Markets

the International Journal of Electronic Commerce & Business Media

[Manuscript Template]

Article Identification Number:	(Only if provided by EM – Electronic Markets)
Full Title of Article:	Performance Analysis and Simulation of Composite Web Services
Subtitle (optional):	
Preferred Abbreviated Title for Running Head:	Chandrasekaran, Miller, Silver, Arpinar and Sheth
Key Words (for indexing and abstract services – up to 6 words):	Web services, Web processes, Web service composition, Simulation, Performance evaluation
Word Count	5689
Word Processing Program Name and Version Number:	Microsoft Word 2002 (10.4219.4219)

Abstract:

The new paradigm for distributed computing over the Internet is that of Web services. They are Web accessible software components, which can be combined and linked together to create new functionality in the form of Web processes. This creates the need to compose services into processes that are efficient in terms of their performance. This paper describes our Service Composition and Execution Tool (SCET) and various methodologies that could be adopted for evaluating the performance of a Web process. SCET allows for composing services statically using its designer and storing them as Web Service Flow Language (WSFL) based specifications. Executing a process enables one to realize its functionality and also analyze its performance. For executing a process, SCET automatically generates execution code for a composed process. This work discusses techniques for process execution time analysis and execution monitoring that can be used to evaluate the performance of individual Web services involved in a process. As processes involving real world services make performance testing difficult simulation is used as an alternate technique for analyzing the efficiency of a process. SCET is integrated with the JSIM simulator, enabling users to simulate a process and get statistical performance estimates.

1. Introduction

A new wave of development based upon the eXtensible Markup Language (XML) has started. One interesting aspect of this development influencing the Web is "Web service" technology, which is a new distributed computing paradigm based on XML. Web services are universally accessible software components deployed on the Web. These software components, which are available as services create an interesting scenario in which efficient Web processes can be created with existing services.

Web services are "self contained, self-describing modular applications that can be published, located, and invoked across the Web" (Tidwell 2001). The Web services architecture models software as individual components available on the Web. Such software components are described by an interface listing the collection of operations that are network accessible through standard XML messaging (Colan 2001). Web services are suitable for integrating e-business applications as they allow for creating loosely coupled distributed systems based on XML messaging protocols (e.g., Simple Object Access Protocol (SOAP) (Kulchenko 2002)).

As individual services are limited in their capability, one may need to compose existing Web services to create new functionality in the form of Web processes. Web service composition is the ability to take existing services (or building blocks) and combine them to form new services. In carrying out this composition task, one should be concerned about the efficiency and the Quality of Service (QoS) that the composed process will exhibit upon its execution. This task of composing services to create efficient Web processes raises the following three issues:

- *Composition of a Web Process:* There are two basic ways to compose a Web process: static and dynamic. In static composition, the services to be composed are chosen at design time, while in a dynamic composition, they are chosen at run-time.
- *Execution of a Composed Web Process:* Executing a process enables one to realize its functionality and also analyze its performance. The central authority pattern and peer-to-peer enactment patterns (Benatallah et al. 2002) are two major execution techniques applied for service compositions.
- *Efficiency of a Composed Web Process:* A composed process should be efficient in terms of its service time and its ability to handle higher loads. For static compositions, as the services in the process are bound at design time, the designer can search for services that have operational metrics (such as service time, load capacity, cost and reliability) satisfying the requirements of the problem being solved.

The focus of this paper is on issue number three. The operational metrics of services/processes can be analyzed using a suitable Quality of Service (QoS) model (Cardoso, Miller et al. 2002; Cardoso, Sheth et al. 2002; Miller et al. 2002). Another alternative for estimating the QoS of a process is to utilize simulation analysis (Miller et al. 2002). Simulation can play an important role in evaluating the quality of a Web process, before its actual execution. For dynamic compositions, the actual efficiency of the process cannot be determined until it is invoked. In both static and dynamic compositions, the performance data from the executed process could be analyzed to provide feedback on the efficiency of the composed process as depicted in Figure 1.

In this paper, we address the above issues related to composing efficient processes and executing/monitoring them, using our Service Composition and Execution Tool (SCET). We further describe the methodologies that could be followed in analyzing the performance of individual Web services involved in a process. Processes involving world-altering services and external services make performance testing difficult, as it might be an expensive or impractical task to test them under various conditions. To address this problem, simulation is used as an alternate means to estimate the efficiency of a process. The JSIM simulator (Nair et al. 1996; Miller et al. 1997) integrated with SCET can simulate the execution of a Web process under various hypothetical conditions and generate statistical results. These results approximate the actual invocation, allowing decisions to be made on the behavior of the process without actual execution. This also allows us to find bottlenecks and performance problems in the service components, suggesting reordering or replacement of those components.

The rest of the paper is organized as follows. In Section 2, we review relevant Web service composition issues and discuss related work. Section 3 covers our system architecture, our Service Composition and Execution Tool (SCET), and our process execution technique. Section 4 explains our approach for analyzing the performance efficiency of a composed Web process and evaluating/comparing the invoked Web services. Simulation and its application to Web process composition task are also discussed in this section. Conclusion and future work are presented in Section 5.

2. Web Service Composition

Web service composition is currently an active area of research, with many languages being proposed by academic and industrial research groups. IBM's Web Service Flow Language (WSFL) (Leymann 2001) and

Microsoft's XLANG (Thatte 2001) were two of the earliest languages to define standards for Web services composition. Both languages extended W3C's Web Service Description Language (WSDL) (Christensen 2001), which is the standard language used for describing the syntactic aspects of a Web service. Business Process Execution Language for Web Services (BPEL4WS) (Curbera et al. 2002) is a recently proposed specification that represents the merging of WSFL and XLANG. BPEL4WS combines the graph oriented process representation of WSFL and the structural construct based processes of XLANG into a unified standard for Web services composition. In contrast to these commercial XML based standards, researchers are developing a unique Web service composition language called DAML-S (Ankolekar et al. 2002), which provides for a richer (more semantic) description of Web service compositions.

Web service composition can be done either statically or dynamically (Benatallah et al. 2002). The decision of whether to make a static or dynamic composition depends on the type of process being composed. If the process to be composed is of a fixed nature wherein the business partners/alliances and their service components slowly change, static composition will suffice. On the other hand, if the process has a loosely defined set of functions to perform, or if it has to dynamically adapt to unpredictable changes in the environment, then static composition may be too restrictive. This is because changes in statically composed systems have to be done manually, interrupting the operation of the process. Dynamic composition involves run-time searching of service registries to discover suitable services. The problem of dynamic service composition has recently gained more importance with the emergence of Web services. As the number of service providers and services is always on the rise, dynamic service composition systems such as eFlow (Casati et al. 2000), can take advantage of the availability of a wide variety of services. eFlow allows nodes (activities) to have service selection rules. When the eFlow engine tries to execute an activity it calls a "service broker" which executes the service selection rules and returns a list of services (with ranking information).

Dynamic discovery of Web services and composition of Web processes closely benefit from each other. UDDI, the widely used XML-based registry for advertising businesses and services, utilizes keyword and classification based searching of services and businesses. Paolucci et al. (2002) stress the importance of discovering Web services based on semantic matching between the declarative description of the service being sought and the description of the service being offered. In this direction, Cardoso and Sheth (2002) present a methodology and a set of algorithms for Web service discovery based on three dimensions: syntax, operational metrics and semantics. The SWORD project (Shankar and Fox 2002) is exploring techniques for composing services using logical rules (precondition => post condition) to express the inputs and outputs associated with services. A rule-based expert system is used to automatically determine whether a process could be realized with the given services. It also returns a process plan that realizes the composition.

Fensel and Bussler (2002) have proposed a Web Service Modeling Framework (WSMF) that provides a rich conceptual model for the development and description of Web services/processes. WSMF consists of four main elements: *ontology* (*a*n ontology is a formal, explicit specification of a shared conceptualization (Gruber 1993)), which is a key enabling technology for the semantic Web (Berners-Lee et al. 2001), *goal repositories* that define the problems that should be solved by Web services, *Web services descriptions* that define the various aspects of Web services and *mediators* which resolve interoperability problems among different services.

Run-time adaptability of a composed process is another research issue in this area. Earlier work on the METEOR (Kochut et al. 1999) project, at the LSDIS lab of the University of Georgia, has addressed this issue for workflows. The distributed task schedulers of METEOR perform the work of scheduling the execution of tasks. This allows for dynamic information to be provided to these task schedulers at run-time, creating an adaptive workflow.

Testing and simulating Web processes for performance evaluation are two new areas with little previous work. Testing Web services for performance evaluation has been classified into load testing, stress testing and spike testing (Daniel and Virgilio, 2001). Simulation of composite Web services is analogous to simulation of workflow models (Miller et al. 1995; Miller et al. 2002). The work in simulation that most closely relates to ours is described by Narayanan et al. (2002). In their work, DAML-S service descriptions of composite services are translated to Petri Nets, providing decision procedures for Web service simulation, verification and composition.

3. System Architecture and Implementation

In this section, we introduce the architecture of our system for composing and executing Web services and explain it with an example scenario. Figure 2 shows the system architecture of our composition and execution tool. The main modules in our system are the Service Composition and Execution Tool (SCET), the JSIM simulator and the Perl execution controller. SCET in turn has a process designer, simulation model generator, Perl code generator and execution monitor sub-modules to help the composer to easily compose, simulate, execute and monitor a process, respectively. The primary sub-modules of SCET are discussed in subsequent subsections.

Central to the architecture is the SCET's Process Designer, a graphical design tool allowing users to statically compose processes. Users can design a digraph representing the process and specify the necessary information associated with the nodes and the links of the graph. The designer can store the composed process as an XML document stored either in a file or in a repository (e.g., using the Db4XML native XML database system (Sipani et al. 2002)). As explained earlier, WSFL, XLANG, BPEL4WS and DAML-S are among the languages that can specify composition of Web services. WSFL was chosen as the language for specifying composed processes in our system. We have extended WSFL's specification to include time, cost and reliability QoS attributes for the activity nodes in the process description (Silver et al. 2003). WSFL precedes BPEL4WS, which was proposed after the completion of this project, but much of the contributions of our work remain.

The JSIM simulator used in our system requires a Java based specification of the model that is to be simulated. In SCET, we represent a composed process as a WSFL based specification. Thus, we need to convert this WSFL based process specification to a model, which the JSIM simulator can interpret. The Simulation Model Generator of SCET automatically transforms the WSFL based process designs into JSIM simulation models.

The Execution Code Generator is capable of generating Perl execution code for the composed process, allowing for straightforward execution. The Execution Code Generator traverses the WSFL specification of the process and generates a Perl program, with code blocks that correspond to the elements encountered in the specification file. For example, an activity node is transformed into a Perl Web service invocation code block as shown in Figure 5.

3.1 Scenario

Figure 3 depicts the internal tasks involved in processing a customer's book purchase order ("BarnesBookPurchase" service). The activities in this process are BarnesGetPrice, CheckCredit, CheckInventory, GenerateBackOrder, ReleaseOrder and SendCreditLowInfo. The bookstore has a real Web service associated with each of the activities involved in this process. These Web services are composed to create a customer order handling Web process. The price of the book chosen by the customer is retrieved using the BarnesGetPrice service. The user's account is then checked for sufficient funds using the CheckCredit service. CheckCredit is an example of an XOR split activity. After the CheckCredit service, the control flows in one of the two outgoing links depending on whether it returns success or failure. If the user has sufficient credit, the CheckInventory service is invoked; otherwise, the SendCreditLowInfo service is invoked. If the CheckInventory Web service returns true the ReleaseOrder service is invoked to send the books; otherwise, the GenerateBackOrder service is invoked.

3.2 SCET Process Composition

In our system, a Web process is represented as a digraph using the Process designer's source nodes, sink nodes, activity nodes, data links and control links (Figure 3). The black transition links in the figure represent control/data links, while the green transition links (not in the picture) represent the data links between the activities. This representation, which is also frequently used in Workflow Management Systems (Sheth et al. 1996), models a process in terms of activity nodes, control links and data links. The activity nodes in the digraph represent the services/tasks in the Web process. The control links specify the control flow within the process. They also capture various constructs such as XOR-splits, AND-splits, XOR-joins, and AND-joins that are normally associated with process designs. For example, an XOR-split represents a node in the process, where based on the conditions the control can flow in only one of the several outgoing links. The data links in a digraph represent the data that flows between two activities and also the information about any mapping that needs to be applied between the output of one activity and the input of another activity.

The process composer apart from laying out the process structure also provides information about the activities (as shown in the Activity Definition dialog box in Figure 3) and the links used in the process. An activity node stores information about the Web service implementing it. This includes the Web service's WSDL file location, the operation being invoked, and QoS information (such as mean service time, reliability factor, cost associated with the activity, etc.). The QoS information, which could be obtained either from the service provider or by performing analysis tests (as explained in the next section), is used to simulate the process behavior. For control links, the composer specifies the condition under which the control will flow along that link, while for data links, the composer specifies how the output of one activity is routed to the input of another activity.

In choosing Web services for a process, one has to consider how the chosen Web services will inter-operate with each other, with respect to the data mappings that need to be applied between the output of one service and the input of another service. For static compositions, the process composer can add the necessary logic to his process definition to handle data mappings between the predefined Web services. This program logic serves as an intermediary/adapter performing tasks such as extracting relevant information or changing the structure of a Web service's output, to make it compatible with the subsequent service's input requirements.

3.3 WSFL Based Specification Generation

SCET saves the process composition as a WSFL based specification. The transformation from the internal storage model to WSFL specification is straightforward. The source/sink nodes in the process design result in source/sink elements of the WSFL specification. The activity nodes of the design are converted to the activity elements of the WSFL specification. The input/output information associated with each activity is transformed into input/output message elements of the corresponding activities in WSFL. The links connecting the nodes in the digraph generate either control link or data link elements based on the type of link used by the composer while designing the process. Figure 4 shows a code fragment of the WSFL based specification generated for the "BarnesBookPurchase" process composition. In the next section, we discuss the Web process execution approach that has been followed in SCET.

3.4 Web Process Execution

A Web process execution is similar to a workflow enactment, the difference being that the components of a workflow are activities while the components of a Web process are services. Web services differ from workflow activities in their distribution, autonomy and heterogeneity. Substantial research on workflow enactment has been done in the Large Scale Distributed Information Systems Lab (LSDIS) at the University of Georgia (Sheth et al. 1996; Miller et al. 1996; Miller et al. 1998; and Kochut et al. 1999). Both centralized and distributed enactment engines were developed as part of the METEOR project (Kochut et al. 1999; Miller et al. 1998; and Sheth et al. 1996).

A composed Web process can be executed either via a centralized approach or a distributed approach. The centralized approach is based on the client/server architecture, with a scheduler, which controls the execution of the components of the Web process. The controller (client) can reside either in the host where the composition is made or at a separate host to allow for better load balancing. The controller/scheduler invokes a Web service, gets the results, and based on the results and the Web process design specification, the controller then invokes the next appropriate Web service. This is the easiest approach for executing Web processes. eFlow (Casati et al. 2001) is such a system with a centralized execution engine.

The distributed approach for Web process execution is more complex. In a distributed approach, the Web services involved are expected to collaborate and share the execution context to realize the distributed execution. In this approach, each Web service involved hosts a coordinator component, which collaborates with other coordinators to realize the execution (Benatallah et al. 2001, 2002). A slightly modified version of distributed execution involves coordinators, which control a set of Web services. The process is executed in a distributed fashion by these coordinators, but internally each coordinator implements a centralized model for executing the tasks controlled by it (Benatallah et al. 2001).

We have followed the centralized execution approach in the initial implementation of our system. SCET is capable of automatically generating Perl execution code from WSFL based process descriptions. In our implementation, a centralized controller manages the entire Web process execution. The centralized execution controller interconnects the activities using a pipe-and-filter model, wherein the output of one Web service is appropriately routed to the input of another Web service based on the control flow and data mappings that were specified by the designer.

There are many alternatives for choosing a language for executing Web processes. Java, C#, Perl, Python and Ruby were among the languages considered for our system. We chose Perl as our execution language, because its easy to use SOAP::Lite (Kulchenko 2002) modules help in quickly scripting the process from its WSFL description. Java is another viable option, which does include powerful execution features, but the execution code size of Java is bulkier than that of Perl (see Silver et al. 2003 for a comparison).

Perl allows us to capture the execution logic of the process including AND/XOR-splits and AND/XOR-joins. We have tested the implementation of these constructs in Perl using its process management utilities: fork() and wait(). Forking a process helps realize parallel execution for AND-splits. For joins in a process, a Perl process (representing a Web service invocation) can wait for other processes, thereby allowing one activity to synchronize with other activities. Perl's thread management features could also help realize the above-mentioned constructs.

As invocation of a Web service via Perl is simple, the transformation of WSFL process specification into Perl execution code, involves converting the WSFL constructs to Perl code blocks. In the "BarnesBookPurchase" Web process, we invoke the getPrice() method on the BarnesGetPrice Web service with bookIsbn as its string parameter. This Web service invocation is converted to the following simple Perl snippet (Figure 5), which is capable of realizing the BarnesGetPrice activity of the process and storing its result for further usage. In the next section, we discuss techniques for evaluating the performance of a composed process.

4. Performance Evaluation

Performance evaluation of Web services can help implementers understand the behavior of the activities in a composed process. Since the performance of a single Web service has the potential to affect the performance of an entire Web process, it is wise to evaluate the performance of the services within a process before making it available for commercial usage. "The most commonly used approach to obtain performance results of a given Web service is performance testing, which means to run tests to determine the performance of the service under specific application and workload conditions" (Daniel et al. 2001). From the user's (process composer's) perspective, the total execution time of a process is a measure of its efficiency.

This work represents a first in developing our overall Quality of Service (QoS) modeling and analysis procedure. Cardoso, Miller et al. (2002) have developed a comprehensive model for the specification of QoS of workflows and Web processes. They have investigated dimensions such as time, cost, reliability and fidelity required to develop a usable QoS model.

4.1 Time Analysis

The execution time taken by a single Web service invocation has three components: Service Time (*S*), Message Delay Time (*M*) and Waiting Time (*W*) (Cardoso and Sheth 2002; Chandrasekaran et al. 2002). Service Time is the time that the Web service takes to perform its task. Message Delay Time is the time taken to send/receive SOAP messages by the invocation call. It is determined by the size of the SOAP message transmitted/received and the load on the network through which the message is being sent/received. Waiting Time is the Web service invocation delay caused by the load on the system where the Web service is deployed. Thus, the Total Invocation Time (*T*) for a Web service σ is given by the following formula.

$$T(\sigma) = M(\sigma) + W(\sigma) + S(\sigma)$$

Evaluating the above three components of *T* for a Web service invocation, will help in analyzing the efficiency of a Web process. We have performed tests to determine each of the above three components for all the Web service invocations used in the process. Message Delay Time was estimated by invoking a ping function for each Web service. XML messages were sent and received, but the Web service performed no work. Service Time was estimated by running tests against the Web service in an environment where the load and waiting delay for the service were controlled. Waiting Time was estimated by running the test in an environment where the Web service was loaded with requests.

Figure 6 shows a snapshot of the sample test result for the example Web process. The time values in the figure are the sum of the Service Time (*S*) and the Message Delay Time (*M*), as the hosts were controlled and the services were not loaded during the test. The *BarnesGetPrice* service used in our example is a Web service hosted by Xmethods (Hong et al. 2000), while the other Web services were hosted locally. The WSDL files for the services used in the above test are available at, *www.xmethods.net/sd/2001/BNQuoteService.wsdl* and *www.cs.uga.edu/~jam/sent/*.wsdl*.

Similar tests are performed when the Web services are loaded to determine the three time measures (Service Time, Message Delay Time, Waiting Time) (Figure 7) for each service invocation. This provides information for analyzing the performance of individual Web services being used in the composed process. In Figure 7, which shows the distribution of the overall time for one of our tests, the *SendLowCreditInfo* Web service has a high Waiting Time. This may indicate that either the Web service or the system hosting that service is not able to handle the load. Replacing the *SendLowCreditInfo* Web service with an equivalent service, which can handle more load, may improve the quality of service of the entire Web process.

The above work on practically analyzing the time dimension of a Web service's QoS model serves as a starting point to realize the mathematical models of Cardoso, Miller et al. (2002). The individual time components estimated using our approach could be used in their model to compute the overall time estimate for the entire process. The algorithm implemented in their work, could also be used to estimate the overall Service Time (*S*), Waiting Time (*W*) and Message Delay Time (*M*) for the entire Web process, providing more information on the QoS of the process.

Load Testing is another approach that can be used to measure the performance of a Web service. The Web services are gradually loaded with client invocations and the performance results are measured. After a certain load point the performance of the Web service will start degrading. This point is the load range to which the Web service is performing effectively. This testing is a useful means of comparing and determining which Web service to choose for a process. Figure 8 shows the time taken by the Web service requests with respect to the number of simultaneous load requests.

4.2 Monitoring Processes

SCET is capable of monitoring a process that is being executed. It can visually show the number of Web service invocations present at the host of the service provider. The number of Web service invocations present at the host for Web service σ for the n^{th} invocation of σ can be guesstimated by the following formula.

$$L_n(\sigma) \quad = \quad [\, W_n(\sigma) + S_n(\sigma) \,] \;/\; S_{MA}(\sigma)$$

where $S_n(\sigma)$, $W_n(\sigma)$ and $S_{MA}(\sigma)$ are the n^{th} service time, the n^{th} waiting time and the moving average of the service times, respectively. The moving average of the service times is computing by keeping a running sum of all n service times and dividing it by the n.

The yellow queue bar in each activity of Figure 3 indicates this estimate of the number of invocations queued for each Web service. During execution of a Web process, the size of the queue varies according to the load on each Web service. The SCET designer is linked with the centralized execution controller to realize this.

We have used the Java RMI mechanism for realizing the execution monitoring functionality in our system. The centralized execution controller executes the process and obtains the monitoring data. This data is then passed to the designer through a Java RMI client. The Java RMI client sends the monitoring data to the Java RMI server of the SCET designer to allow users to visually monitor the process execution. We have adopted this approach in our system as a bridge to communicate information between the Perl based centralized execution controller and our Java based SCET designer.

The performance analysis techniques described above provide feedback on the quality of the composed processes. These techniques also have some associated disadvantages. They can yield good estimates if the Web services involved are under our control. This is because all these tests need to be executed in a controlled manner, taking care of the load on the system when the testing is done. The distribution and autonomy of Web services makes meeting this requirement difficult. If there is a Web service over which we do not have control, then it is difficult to get accurate performance testing results. The server hosting the Web service may be heavily loaded with other programs making it hard to obtain accurate estimates of service times.

Further, if the services involved in the process include "world-altering" services (such as flight-booking service, money transfer service) or if there is a cost involved in invoking the individual services, then analyzing the performance of the process by executing it may not be feasible. To overcome these difficulties, simulation based testing, which is described in the next subsection could be used as an alternative technique for evaluating the process.

4.3. Simulation

Simulation helps in determining how the composed Web services will perform when they are deployed. It plays an important role by exploring the “what-if” questions during the process composition phase, which may not be feasible or too costly to explore via performance tests with real Web processes. Simulation can provide feedback on the process that was composed allowing the composer to modify his/her process design by

- replacing services that do not provide required service time averages,
- modifying the process structure (such as altering the number of activities involved, and changing the control flow) based on the simulation runs.

When testing is difficult or costly, simulation is a useful approach. Once calibrated from real data or reasonable guestimates, the simulation may interpolate/extrapolate performance or explore the effects of replacing Web services before actually doing so.

SCET is integrated with the latest version of JSIM, a Java-based simulation and animation environment (Nair et al. 1996; Miller et al. 1997) that contains several features to support simulation of Web processes. JSIM currently simulates both the centralized execution of a Web process (Figure 9), as well as the distributed version see the (Silver et al. 2003). In this paper, we simulate the centralized representation, which requires all Web services to communicate with the central controller. JSIM’s server/faculty nodes serve as the activities of a Web process. JSIM’s source node, sink node and transports map to source node, sink node and control links of the Web process. The JSIM simulation entities represent the Web process invocations. Chandrasekaran et al. (2002) elaborate more on this mapping between JSIM and a Web process.

After composing the process using the designer, the composer can make use of the simulation model generator to convert the WSFL process specification to a JSIM simulation model. The JSIM simulation model takes as input the service time distribution functions characterizing the Web services. For each of the control links involved in the process, JSIM requires an associated probability value for simulating the process execution. In our experiments, we have computed the probability values associated with each of the control links, by executing the process on a test basis.

After its simulation run, JSIM generates statistical information (Figure 10) about the completed simulation. This includes information about minimum, maximum, mean, and standard deviation of the time estimates of each of the activities involved in the process model. During this simulation phase, the composer can analyze how the process and individual Web services will perform when the process structure is changed, or when Web services with better service time averages are being used in the process. The statistical information generated for these test runs provides feedback on the process performance for hypothetical cases. This creates a feedback loop when composing Web services, thereby allowing the composer to iterate through this feedback loop (as shown in Figure 1) until an efficient process suitable for execution is obtained.

The JSIM Simulation displays dynamically the number of entities (invocations) being queued up in an activity. The SCET designer is also capable of displaying the estimated number of real invocations present in the host of a Web service. Thus, we can do both simulation and execution and can compare the two models quantitatively (Figure 10, Figure 6), as well as visually, to help determine the validity of the simulation model. If the simulation model is found not to be valid, we can make changes to the simulation parameters to improve its accuracy.

In the comparison test made above, the actual process was executed several times with a three second time period between successive process executions. The corresponding simulation of the process generated several simulation entities with an inter-arrival time of three seconds. Our preliminary simulation results approximate the actual execution values.

5. Conclusions and Future Work

Web services composition is a new research area that combines Web services technology and process composition. In this paper, we have focused on problems related to composition representation, specification of service compositions and execution of processes, using our Service Composition and Execution Tool (SCET). We have described the key components of SCET and how it can be used to compose efficient Web processes. SCET allows users to statically compose services to form processes and store them as WSFL based specifications. As QoS specification is an integral part of describing a process, we have enhanced WSFL to include QoS attributes for specifying requirements on quality such as time, cost and reliability. SCET executes a process by following a centralized process execution technique. Services should be composed taking into consideration the overall performance efficiency of the resulting process. In this direction, we present performance analysis approaches that help a composer to evaluate the performance of a process. We also describe how simulation can be used with a Web process composition system to carry out efficient compositions.

However, there are still several issues that need to be addressed. First, SCET is a static composition tool. SCET needs to support dynamic composition, so that users can just specify their requirements on the services and not the actual service. As a first step in this direction, we are adding a service discovery capability to SCET. When choosing an initial Web service or a replacement service, a user may pose high-level queries to one or more UDDI registries to retrieve candidate Web services that the user may choose from. Currently, SCET supports WSFL based process specification. BPEL4WS is a newly proposed composition standard for Web services. SCET needs to be enhanced to support this new standard. SCET also needs changes to its framework, to accommodate for this kind of evolving nature of process specification languages. From the execution point of view, the execution code generator has restrictions on its functionality. It is capable of handling services which return primitive data types. This needs to be improved to support array values and objects. Also, we need to address the issue of data mapping between Web services, so that users can specify their data transformation function to map information between Web services. In this work, we have provided approaches for realizing the time QoS dimension of a Web process. We would need to enhance it to address other QoS dimensions such as reliability and cost.

References

Ankolekar, A., Burstein, M., Hobbs, J., Lassila, O., Martin, D., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Payne, T., and Sycara, K. (2002) 'DAML-S: Web Service Description for the Semantic Web', in *International Semantic Web Conference*, Sardinia, Italy, pp. 348-363.

Benatallah, B., Dumas, M., Fauvet, M., and Paik, H. (2001) 'Self-Coordinated and Self-Traced Composite Services with Dynamic Provider Selection', Technical report, School of Computer Science & Engineering, The University of New South Wales.

Benatallah, B., Dumas, M., Fauvet, M., and Rabhi, F. (2002) ‘Towards Patterns of Web Services Composition', in *Patterns and Skeletons for Parallel and Distributed Computing,* Springer Verlag, UK (to appear).

Berners-Lee, T., Handler, J., and Lassila, O. (2001) 'The Semantic Web', http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21, accessed 22 September 2002.

Cardoso, J., Miller, J., Sheth, A., and Arnold, J. (2002) 'Modeling Quality of Service for Workflows and Web Service Processes', Technical Report, LSDIS Lab, Computer Science Department, The University of Georgia.

Cardoso, J. and Sheth, A. (2002) ‘Semantic e-Workflow Composition', Technical Report, LSDIS Lab, Computer Science Department, The University of Georgia.

Cardoso, J., Sheth, A., and Miller, J. (2002) 'Workflow Quality of Service', Enterprise Inter- and Intra-Organizational Integration – Building International Consensus, Kluwer Academic Publishers, pp. 303-312.

Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V., and Shan, M. (2000) 'Adaptive and Dynamic Service Composition in eFlow', in *Proceedings of the International Conference on Advanced Information Systems Engineering,* Stockholm, Sweden, pp. 13-31.

Chandrasekaran, S., Silver, G., Miller, J., Cardoso, J., and Sheth, A. (2002) 'Web Service Technologies and their Synergy with Simulation' in *Proceedings of the 2002 Winter Simulation Conference, San* Diego, CA, pp. 606-615.

Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. (2001) 'Web Services Description Language (WSDL) 1.1.', http://www.w3c.org/TR/wsdl, accessed 22 September 2002.

Colan, M. (2001) 'An Overview of Web Services', http://www-106.ibm.com/developerworks/Webservices, accessed 22 September 2002.

Curbera, F., Goland, Y., Klein, J., Leymann, F., Roller, D., Thatte, S., and Weerawarana, S. (2002) 'Business Process Execution Language for Web Services', http://msdn.microsoft.com/Webservices/default.asp?pull=/library/en-us/dnbiz2k2/html/bpel1-0.asp, accessed 22 September 2002.

Daniel, A. and Virgilio, A. (2001) *Capacity Planning for Web Services: Metrics, Models, and Methods*, Prentice Hall, Englewood Cliffs, NJ.

Fensel, D. and Bussler, C. (2002) 'The Web Service Modeling Framework WSMF', http://www.cs.vu.nl/~dieter/ftp/paper/wsmf.pdf, accessed 22 September 2002.

Gruber, T. (1993) 'Towards Principles of the Design of Ontologies used for Knowledge Sharing', Technical Report KSL-93-04, Knowledge Systems Laboratory, Stanford University.

Hong, T. and Hong, J. (2000) ‘X Methods’, http://www.xmethods.net, accessed 22 September 2002.

Kochut, K., Sheth, A., and Miller, J. (1999) 'Optimizing Workflow', in *Component Strategies*, Vol.1, No. 9, pp. 45-57 (SIGS Publications Inc).

Kulchenko, P. (2002) 'SOAP::Lite for Perl', http://www.soaplite.com, accessed 22 September 2002.

Leymann, F.(2001) 'Web service flow language (WSFL) 1.0', http://www-4.ibm.com/software/solutions/Webservices/pdf/WSFL.pdf, accessed 22 September 2002.

Miller, J., Cardoso, J., and Silver, G. (2002) 'Using Simulation to Facilitate Effective Workflow Adaptation', in *Proceedings of 35th Annual Simulation Symposium*, San Diego, CA, pp.177-181.

Miller, J., Nair, R., Zhang, Z., and Zhao, H. (1997) 'JSIM: A Java-based Simulation and Animation Environment', in *Proceedings of the 30th Annual Simulation Symposium*, Atlanta, GA, pp. 31-42.

Miller, J., Palaniswami, D., Sheth, A., Kochut, K., and Singh, H. (1998) 'WebWork: METEOR's Web-based Workflow Management System', in *Journal of Intelligent Information Systems*, Vol. 10-2, pp. 185-215.

Miller, J., Sheth, A., Kochut, K., Wang, X., and Murugan A. (1995) 'Simulation Modeling within Workflow Technology', in *Proceedings of the 1995 Winter Simulation Conference*, Arlington, Virginia, pp. 612-619.

Miller, J., Sheth, A., Kochut, K., and Wang, X. (1996) 'CORBA-Based Run-Time Architectures for Workflow Management Systems' in *Journal of Database Management*, Special Issue on Multidatabases, Vol. 7-1, pp. 16-27.

Nair, R., Miller, J., and Zhang, Z. (1996) 'JSIM: A Java-based Query Driven Simulation Environment', in *Proceedings of the 1996 Winter Simulation Conference*, Coronado CA, pp. 786-793.

Narayanan, S. and Mcllraith, S. (2002) 'Simulation, Verification and Automated Composition of Web Services', in *Proceedings of the Eleventh International World Wide Web Conference*, Honolulu, HI, pp. 77-88.

Paolucci, M., Kawamura, T., Payne, T., and Sycara, K. (2002) 'Semantic Matching of Web Services Capabilities', in *Proceedings of the First International Semantic Web Conference,* Sardinia, Italia, pp. 333-347.

Shankar, P. and Fox, A. (2002) 'SWORD: A Developer Toolkit for Web Service Composition', in *Proceedings of the Eleventh International World Wide Web Conference*, Honolulu, HI.

Sheth, A., Kochut, K., Miller, J., Worah, D., Das, S., Lin, C., Lynch, J., Palaniswami, D., and Shevchenko, I. (1996) 'Supporting State-wide Immunization Tracking using Multi-paradigm Workflow Technology', in *Proceedings of the 22nd International Conference on Very Large Databases*, Bombay, India, pp. 263-273.

Silver, G., Miller, J., Sheth, A. (2003), 'Modeling and Simulation of Quality of Service for Composite Web Services', in *Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, FL. (in preparation)

Sipani, S., Verma, K., Chandrasekaran, S., Zeng, X., Zhu, J., Che, D., and Wong, K. (2002) 'Designing an XML Database Engine: API and Performance', in *Proceedings of the 40th Annual Southeast ACM Conference*, Raleigh, NC, pp. 239-245.

Tidwell, D. (2000) 'Web Services – The Web's Next Revolution', http://www-106.ibm.com/developerworks/Webservices, accessed 22 September 2002.

Thatte, S. (2001), 'XLANG: Web Services for Business Process Design', http://www.gotdotnetcom/team/xml_wsspecs/xlang-c/default.htm, accessed 22 September 2002.

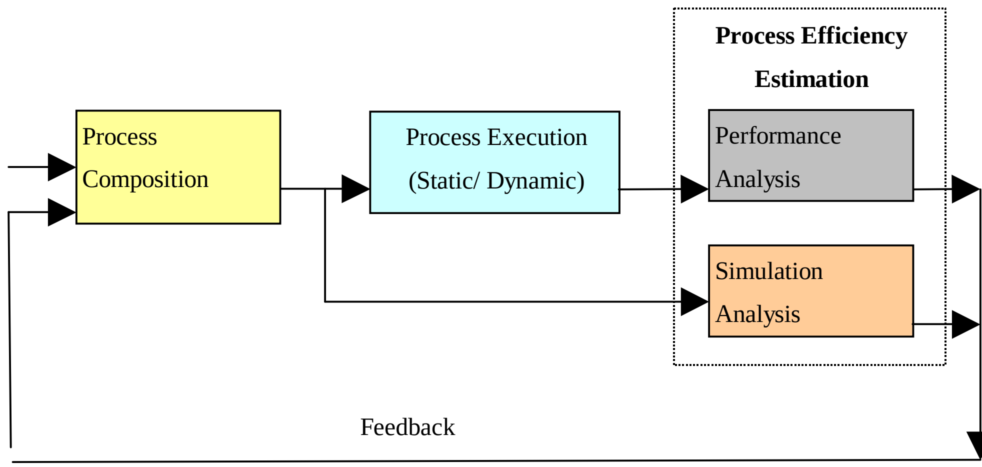

Figure 1. Process Composition Tasks

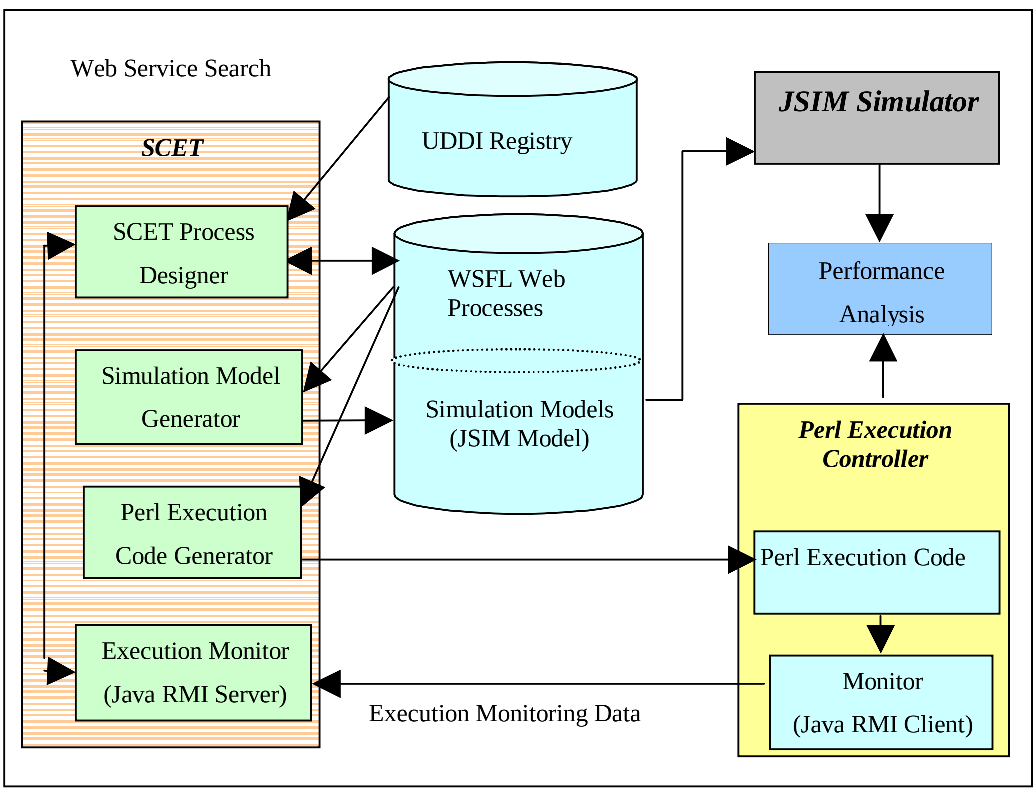

Figure 2. System Architecture for SCET

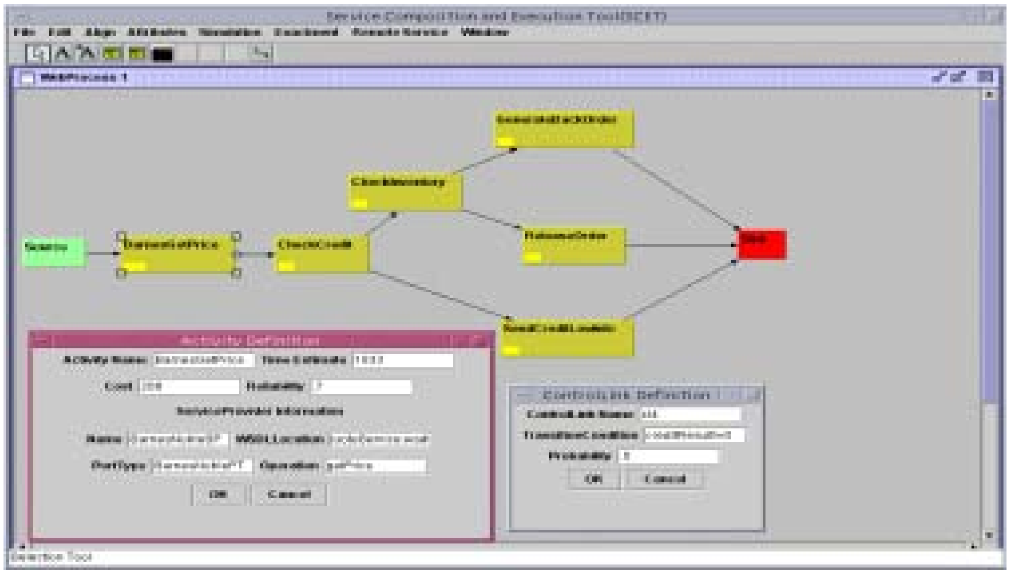

Figure 3. Web Process Composition using SCET

```
<definitions>
<message name="CheckInventoryInput">
                <part name="bookIsbn"  element="string"/>
</message>
......................
......................
<serviceProvider  name="CheckInventorySP">
  <locator
        type="static" service=
                "http://www.cs.uga.edu/sent/xmethods/CheckInventory.wsdl"/>
</serviceProvider>
......................
......................
<activity  name="CheckInventory" X1="336.0" Y1="104.0"
                X2="481.0" Y2="152.0" time="79" cost="200" reliability=".7" type="Facility">
        <input message="CheckInventoryInput"/>
        <output message="CheckInventoryOutput"/>
        <join condition="true"  when="deferred"/>
        <performedBy serviceProvider="CheckInventorySP">
                <implement>
                        <export>
                                <plugLink>
                                <target
                                PortType="CheckInventoryPT"
                                Operation="hasBook"/>
                                </plugLink>
                        </export>
                </implement>
        </performedBy>
</activity>
......................
......................

<controlLink
         name="cl6" source="CheckInventory" target="GenerateBackOrder"
        condition="CheckInventoryResult==0" probability=".2"
         X1="431.0" Y1="104.0" X2="490.0" Y2="72.0" />
......................
......................
<dataLink  name="dl2" X1="238.0" Y1="178.0" X2="336.0" Y2="149.0"
                 source="SearchAmazon" target="CheckInventory">
        <mapInfo sourcePart="bookIsbn" targetPart="bookIsbn"/>
</dataLink>
</definitions>
```

Figure 4. WSFL Process Description

```
my $GetBNPrice = SOAP::Lite
        -> service ('http://www.xmethods.net/sd/2001/BNQuoteService.wsdl');
$GetBNPriceResult = $GetBNPrice->getPrice($isbn);
```

Figure 5. Perl Execution Code Snippet

```
10: Num of Iterations
            :BarnesGetPrice   :CheckCredit      :SendcreditLowInfo
            :1.913394         :1.342986         :0.759598
            :1.714697         :0.682717         :0.486978
            :1.562682         :1.42475          :0.692787
            :1.886854         :0.643781         :0.473613
            :1.828123         :0.816607         :0.525161
            :1.765729         :0.876123         :0.486001
            :1.502503         :0.93461          :0.534121
            :1.617242         :0.644926         :0.464647
            :2.485304         :0.82555          :0.691154
            :2.126191         :0.855064         :0.472463
            :1.84127          :0.918563         :0.57291


Total       :18.330595        :8.622691         :5.399835
Average     :1.8330595        :0.8622691        :0.5399835
Minimum     :1.502503         :0.643781         :0.464647
Maximum     :2.4853           :1.42475          :0.692787
STD         :0.276012         :0.2134661        :0.0824032
```

Figure 6. Test Results for Total Invocation Time (*T*)

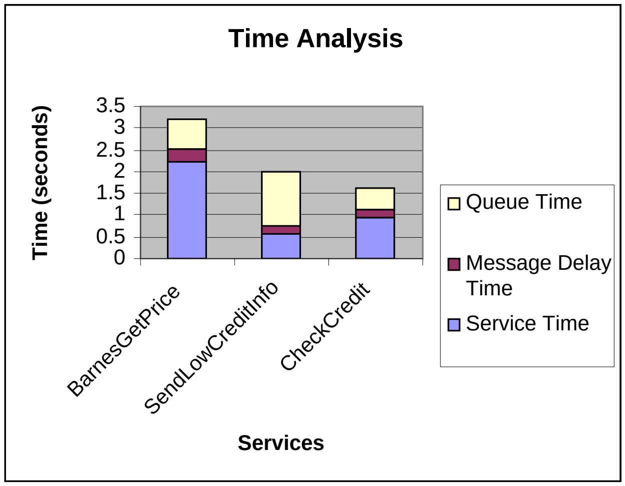

Figure 7. Performance: Time Analysis

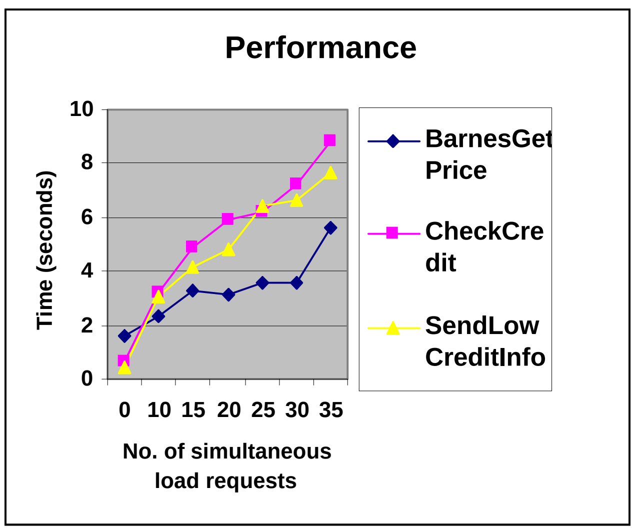

Figure 8. Load Testing Results for the Barnes Book Purchase Web Process

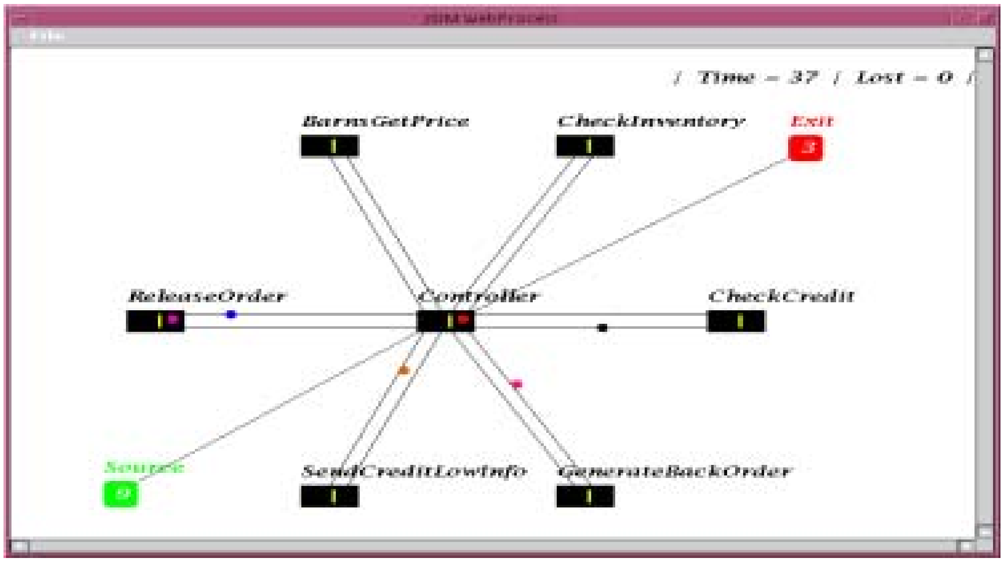

Figure 9. JSIM Centralized Execution Simulation

NoSamples	MinValue	MaxValue	MeanValue	Deviation	Interval	Precision	StatName
10.0	3000.0	3000.0	3000.0	0.0	0.0	0.0	Source (dur)
10.0	0.0	0.0	0.0	0.0	0.0		Exit (dur)
10.0	1832.938	1833.028	1833.002	0.025	0.019	0.0	BarnesGetPrice (du
36033.0	0.0	1.0	0.509	0.5	0.005	0.010	BarnesGetPrice (oc
10.0	861.947	862.058	861.997	0.034	0.026	0.0	CheckCredit (dur)
38496.0	0.0	1.0	0.224	0.417	0.004	0.015	CheckCredit (occ)
0.0	0.0	0.0	0.0	0.0	0.0	0.0	CheckInventory (d
0.0	0.0	0.0	0.0	0.0	0.0	0.0	CheckInventory (o
10.0	538.951	539.032	538.991	0.029	0.022	0.0	SendCreditLowInfo
41035.0	0.0	1.0	0.131	0.338	0.003	0.025	SendCreditLowInfo
0.0	0.0	0.0	0.0	0.0	0.0	0.0	GenerateBackOrde
0.0	0.0	0.0	0.0	0.0	0.0	0.0	GenerateBackOrde
0.0	0.0	0.0	0.0	0.0	0.0	0.0	ReleaseOrder (dur)
0.0	0.0	0.0	0.0	0.0	0.0	0.0	ReleaseOrder (occ)
30.0	400.0	400.0	400.0	0.0	0.0	0.0	path1 (dur)
0.0	0.0	0.0	0.0	0.0	0.0	0.0	path2 (dur)
0.0	0.0	0.0	0.0	0.0	0.0	0.0	path3 (dur)
0.0	0.0	0.0	0.0	0.0	0.0	0.0	path4 (dur)
0.0	0.0	0.0	0.0	0.0	0.0	0.0	path5 (dur)
0.0	0.0	0.0	0.0	0.0	0.0	0.0	path6 (dur)
140.0	400.0	400.0	400.0	0.0	0.0	0.0	path7 (dur)
50.0	400.0	400.0	400.0	0.0	0.0	0.0	path8 (dur)
40.0	400.0	400.000	400.000	0.0	0.0	0.0	path9 (dur)

Figure 10. JSIM Statistics Table